Space ROS: An Open-Source Framework for Space Robotics and Flight Software

Austin B. Probe
Emergent Space Technologies, Inc., Austin, Texas, 78752, United States

S. Will Chambers, Amalaye Oyake
Blue Origin, Kent, Washington, 98032, United States

Matthew Deans, Guillaume Brat, Nick Cramer, Brian Kempa
NASA Ames Research Center, Moffett Field, California, 94035, United States

Brian Roberts
NASA Goddard Space Flight Center, Greenbelt, Maryland, 20771, United States

Kimberly Hambuchen
NASA Johnson Space Center, Houston, Texas, 77058, United States

Space ROS is an open-source spacecraft flight software framework for developing robotic applications for space being developed by NASA, Open Robotics, Blue Origin, and others. It is designed to be platform independent, portable and project independent. Space ROS is a fork of the ROS 2 framework and conforms to the ROS 2 Application Programming Interface (API) that has been hardened to be compatible with the demands of safety-critical space robotics applications. The intent of Space ROS is to provide a robust framework for space robotic applications where ROS 2 applications can be reused with little to no modification enabling the space community to take advantage of the innovation of the ROS community. This will shorten the time for development of novel space robotics capabilities, enable reuse of capabilities between missions, and lower the life-cycle cost of new robotic missions. This paper details the objectives of Space ROS, the motivation for its creation, the approach for its development and validation, as well as initial benchmarking results.

I. Nomenclature

ROS2 = Robot Operating System
NPR = NASA Procedural Requirements
DRA = Design Reference Application
ARC = Ames Research Center
GSFC = Goddard Space Flight Center
DSA = Distributed Space Autonomy

II. Introduction

ROS has enabled unprecedented open-source collaboration on the development of robotics capabilities and collaboration between academic, commercial, and government developers. The original ROS implementation had

drawbacks in architecture and implementation that made it a challenge to use for demanding and rigorous applications. Open Robotics and the ROS community have worked to address these issues with the development of ROS2. Space ROS is a collaboration between NASA, Open Robotics, Blue Origin, and others that seeks to expand the capabilities of ROS2 into a framework that could be deployed for safety-critical applications in space and leverage the innovation from the academic, commercial, and government robotic communities to meet future opportunities and challenges in space robotics. The technical advancement of the reusable Space ROS framework and its availability to the space industry has the potential to greatly reduce the life cycle costs of space-qualified robots. The Space ROS framework seeks to support a variety of robotic systems including rovers, autonomous spacecraft, dexterous manipulators, humanoid robots, and multi-robot systems.

III. Space ROS

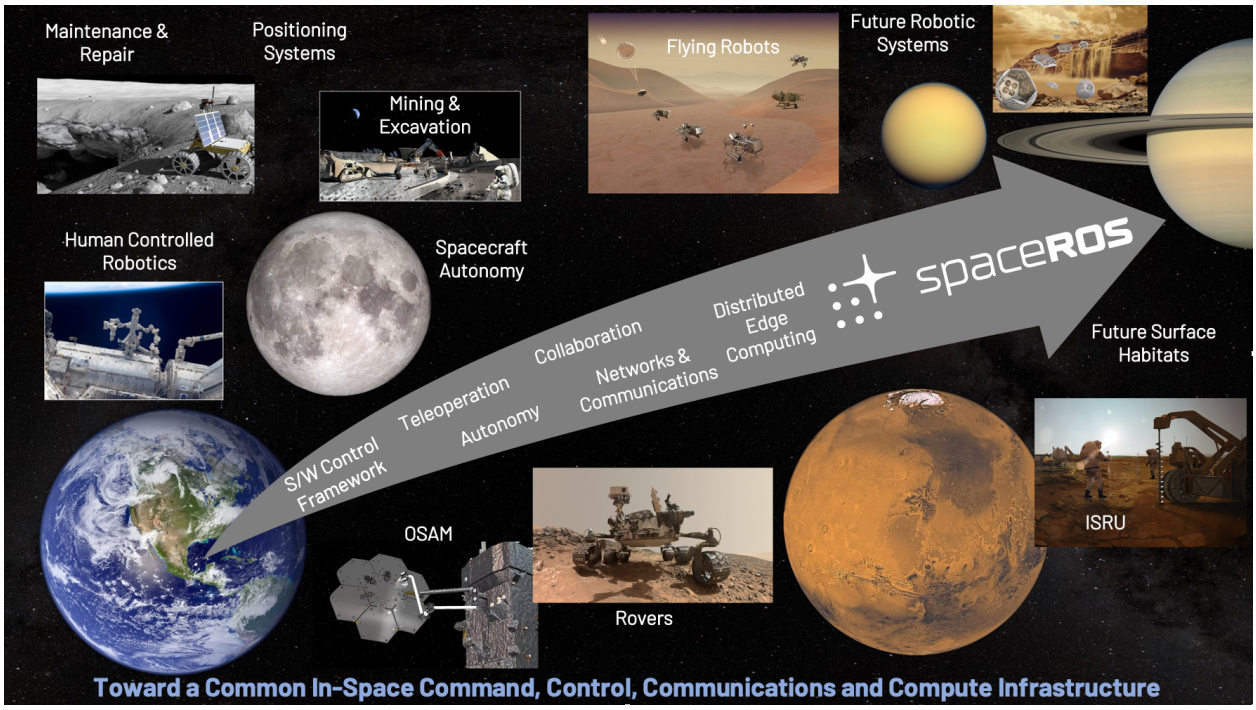

Figure 1: Space ROS Roadmap and Objectives

A. Space ROS Objectives

The Space ROS team is working to develop a certifiable and reusable robotic framework to meet the needs of future space-based robotics applications. This will support many future robotics capabilities and missions as shown in Figure 1. A certifiable system is designed to be aligned with flight software standards, focusing on the NASA NPR 7150.2 as a baseline and so that the framework could be demonstrated to meet or exceed the requirements for use on a Class A Mission as defined by the NASA NPR [1]. This includes addressing: 1) Memory Safety; 2) Deterministic Performance; 3) Static analysis; 4) System Testing and Evaluation. We plan to develop a fork of ROS 2 and address these challenges while maintaining compatibility with the ROS 2 API and enabling ROS 2 applications to be reused on Space ROS with little to no modification enabling the space community to take advantage of the development and innovation of the larger ROS community. The resulting framework will support characteristic space robotics applications while enabling the rapid development of new robotic capabilities and the reuse of capabilities between missions. The Space ROS framework is highly aligned with NASA's published Technology Taxonomy Areas [2], as shown in

Figure 2, demonstrating the need for such a framework to support future space endeavors. Space ROS will provide enabling capabilities addressing needs in Robotics, Autonomy, Guidance, Navigation, and controls as well as supporting technologies like simulation and software modeling.

Figure 2: NASA Technology Taxonomy Areas Addressed with Space ROS

Current mission concepts are using ROS2 as part of their development cycle, such as the Regolith Advanced Surface Systems Operations Robot (RASSOR) project, shown in Figure 3. This generally follows a pattern of implementing a robotic platform using ROS2 for control and as a test bed for algorithmic development and them transitioning the algorithms, once developed and proven effective, to a flight certified software platform such as NASA's cFS once. Developing in ROS2 allows for developers to quickly implement their system leveraging the power of the ROS development community and all the previous development work that has been open sourced. Space ROS would enable missions to deploy FSW for missions without having to perform the second step of converting the algorithms into a second framework, reducing cost and development time.

Figure 3: NASA RASSOR that uses ROS2 for Development

B. Design Reference Applications

To guide the development of Space ROS we have defined a set of Design Reference Applications (DRAs). These are designed to serve as an early assessment of existing (and missing) ROS features for space applications. They will also help identify the driving attributes of Space ROS to achieve "typical" / projected space robotic systems. These DRAs, shown in Figure 4, include Cargo Offloading, Resource Prospecting, Site Preparation, and Orbital Station Unpressurized Robotics. To help guide the development of Space ROS we worked to define a basic description for each DRA, identified and explored their key robotic components and identified technical attributes that a robotic infrastructure supporting that DRA needs to accommodate. The following technical attributes were identified for focus: Perception, Planning, Mobility, Manipulation, Control and Coordination, and Communications. The descriptions as well as a detailed listing of the technical attributes are described below.

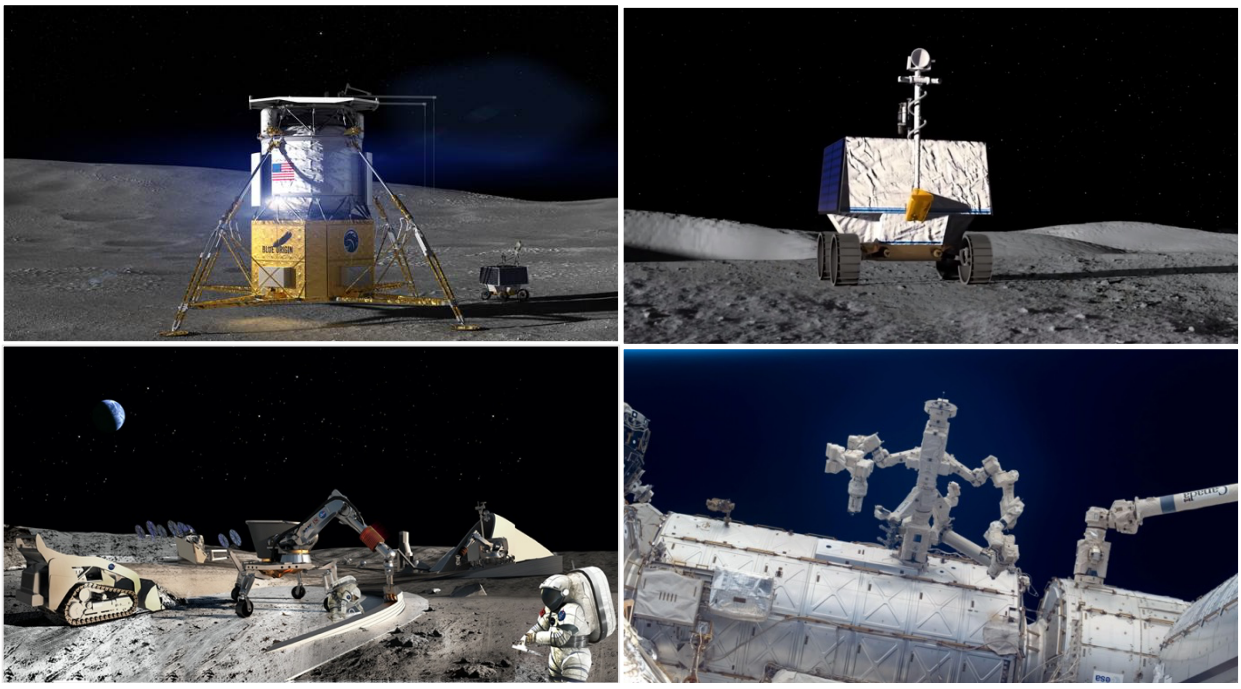

Figure 4: Space ROS Design Reference Applications

1. ***Cargo Offloading***

A robotic crane system on a lander lifting, translating, and depositing cargo from lander deck to surface with mission control in the loop

2. ***Resource Prospecting***

A mobile robotic resource prospector with instruments and a drill performing a survey to characterize regions for amenability for later resource extraction with a data relay and semi-autonomous control

3. ***Site Preparation***

A team of robotic regolith movers and assemblers performing terrain preparation and foundational construction in support of advanced surface operations with remote Earth based supervision

4. ***Orbital Station Unpressurized Robotics***

An external (unpressurized) service robot tasked with operations involving station components and visiting vehicles in support of an orbital space station with local crew member operation

Table 1: Summary of Technical Attributes of Space ROS DRAs

	Cargo Offloading	Resource Prospecting	Site Preparation	Orbital Station Unpressurized Robotics
Perception	• Payload location • Lander structure perception • Surface characterization	• Natural terrain characterization (traversability) in low light (lunar) • Detection of suitable drive paths and sampling sites • Novel feature/signal detection	• Site characterization, including terrain volumetric assessments • Terrain mechanical resistance, mass • Other robot detection	• Proprioception (self-configuration) • Obstacle detection (module hulls, visiting vehicles, external payloads and equipment, astronauts) • Force/moment detection (contact)
Planning	• Manipulator trajectory planning • Payload surface placement planning	• Traversal planning considering remote sensing (resource maps), lighting, communications • Activity planning considering battery state of charge, intended activities and resource costs	• Robot team-based excavation, deposition and leveling planning • Local route planning • Activity planning considering battery state of charge, intended activities and resource costs	• Arm trajectory planning • Task planning
Mobility	None	• Control under rover kinematics and surface terramechanics • Terrain obstacle avoidance • Instrument, manipulator and/or drill placement	• Control under rover kinematics and significant surface interactions • Terrain obstacle avoidance • Tool / manipulator placement and control under motion	• Control under arm kinematics, base contact and payload mass properties • Station obstacle avoidance
Manipulation	• Payload grapple	• Possible active regolith interaction (scooping, scraping)	• Positing and assembly of sit elements	• Closed-chain grappling/initial contact, ungrappling

	• Payload translation under gravity and dynamics • Payload surface placement amongst terrain features	• Drilling / subsurface access	• Possible active regolith interaction (scooping, scraping)	• Free-flyer grappling/initial contact, release • Maneuvering "payloads" under arm kinematics and joint payload/station/arm contact avoidance, force and torque limits • Emergency halt in face of unexpected contact or resistance
Control and Coordination	• Earth-based human supervision • Interleaved lander and robotics tasks (e.g. payload umbilical and mechanical release) • Possible payload health monitoring before/during/after crane operation	• Earth-based human supervision • Rover, instrument, manipulator and/or drill coordinated control •	• Robot team coordination (loose and potentially tight coordination) • Earth-based human supervision •	• Earth-based human supervision • Interleaved lander and robotics tasks (e.g. payload umbilical and mechanical release) • Possible payload health monitoring before/during/after crane operation
Communications	• Crane-Lander: hardline (serial, Ethernet) • Payload-Lander: hardline (via severable umbilical) • Lander-Mission Control: RF	• Prospector-Instruments: hardline (serial, Ethernet) • Prospector-Drill: hardline (serial, Ethernet) • Prospector-Mission Control: RF (direct-to-Earth) • Prospector-Comm Relay: RF • Comm Relay-Mission Control: RF	• Robot-Robot: RF / WiFi • Robot-Mission Control: RF	• Station-Manipulator: hardline • Station-Visiting Vehicle: RF and/or hardline (state-dependent) • Station-Crew Workstation: hardline • Station-ORU: hardline (via standard quick connect/disconnect interface)

C. Community Input

To gauge the community interest in Space ROS and guide its development. The Space ROS team we solicited feedback on the Space ROS concept from the space robotics and autonomy community through a NASA RFI. Responses were received from 30 companies, universities, and individuals from many different countries as shown in Figure 5.

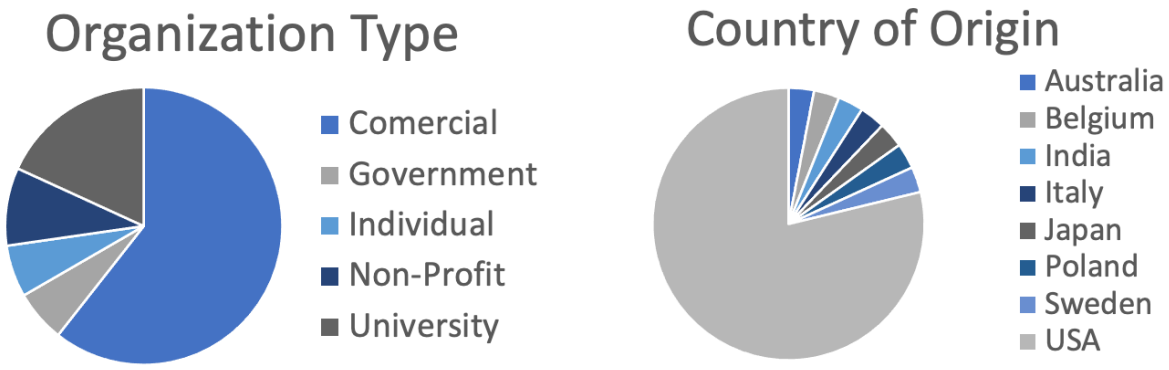

Figure 5: Space ROS RFI Respondent Demographic Summary

Based on the responses to this here is significant interest in a version of ROS2 that can support space systems and robotics. Some key take aways were:

- Operating in a resource constrained environment is critical
- Interactions with embeded systems is a concern for many respondents

- Security is important to organizations working with the DoD
- Documentation and examples are a key to adoption
- A significant portion of respondents are interested in supporting the development community

To augment the technical attributes from the DSAs and guide future Space ROS development the team also requested feedback from industry on their use of ROS/ROS2, their efforts developing robotic systems, and future objectives for those systems. We then anonymized this data and generated statistics based on the responses which are summarized in Figure 6.

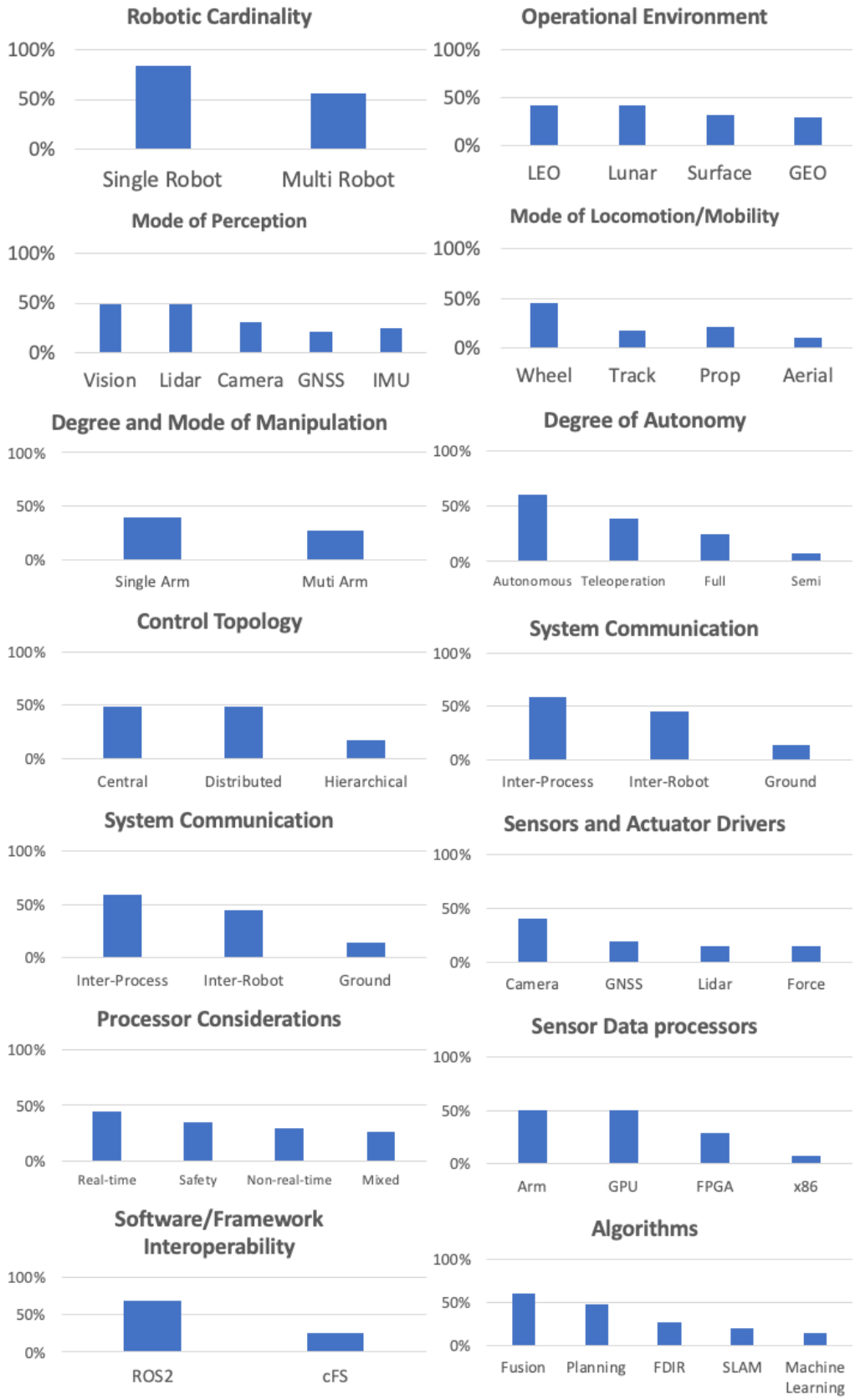

Figure 6: RFI Statistics on Technical Parameters for Robotic Systems

IV. Implementation

A. Safety Approach

The intent of the Space ROS software development effort is to build a workflow that aligns with established safety-critical software standards such as NASA NPR 7150.2 and DO-178C. This will not include flight certification for a particular class of mission but updates to the framework and surrounding process that would accelerate that certification if Space ROS was selected as part of a future mission solution. The current Space ROS framework includes updates to the ROS2 codebase to improve the memory safety and infrastructure tools that support, continuous integration and artifact / document generation for supporting software qualification activities. Finally, plans for Verification and Validation (V&V) have been developed based on previous NASA V&V efforts to lay the groundwork for future development and potential certification efforts.

1. ROS2 Safety Updates

The major update that is required to align ROS2 with safety critical software standards is to improve the memory handling. This was achieved in Space ROS by leveraging a memory pool developed using the polymorphic allocator introduced in the C++ 17 Standard. [3] This allows for an initial allocation of memory that can be used by what would have previously been dynamic memory allocations in the standard ROS2 implementation. This allocation technique currently doesn't apply to the communications middleware, however the ROS2 middleware interface allows for significant middleware modification without impacting the client applications and the ePromisa Fast DDS library that is used as the default ROS2 middleware implementation can be configured to memory preallocation. [4] [5]

2. Infrastructure and Tooling

Open-source efforts like ROS2 have the potential to produce incredibly powerful tools, but there is generally a significant discrepancy between the level of planning and documentation produced and what is expected for safety critical flight qualified software. Therefore, one of the key efforts of the Space ROS effort has been developing the infrastructure and tooling to support development. The objective for the system's pipeline for development is shown in Figure 7. The current version of this pipeline and the surrounding infrastructure that is currently deployed on the Space ROS GitHub repository. This design largely follows the best practices of open source development with additional focus on software requirements and static analysis described in the following paragraphs.

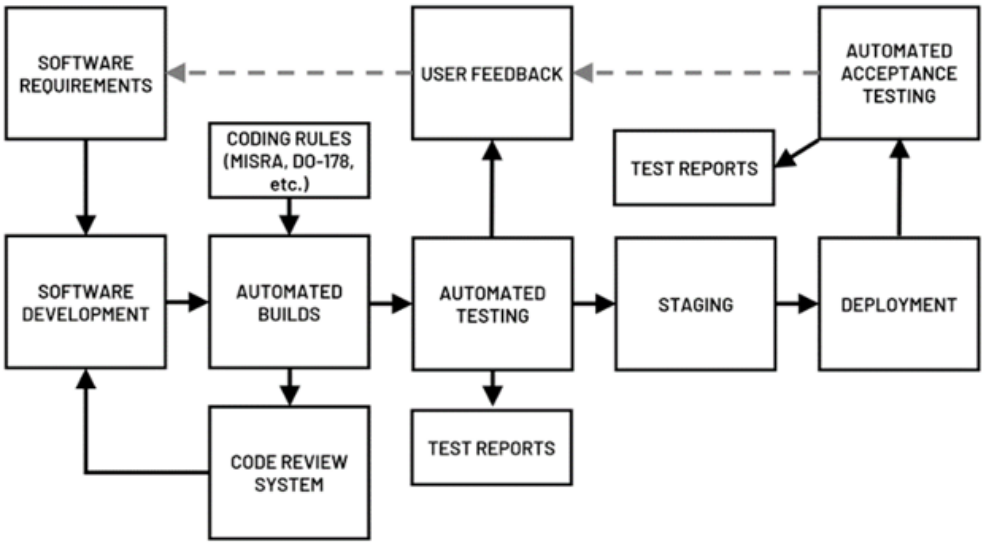

Figure 7: Target Development and Continuous Integration Pipeline for Space ROS

Requirements management in aerospace software development is typically a highly managed process using a strict process and proprietary tools. Requirements for flight software systems must be highly detailed, with multiple levels describing abstract needs to detailed software behavior. This detail is used to support the system V&V and certification so traceability between the requirements and software codebase is frequently desired. In comparison open source development rarely has strictly maintained formal requirements and those that are generated frequently become out of date. The processes of developing and maintaining requirements is often considered "heavyweight" and because open source efforts rely on soliciting contributions from the community processes that make contributing more cumbersome are frequently counterproductive. Additionally, processes that require property software that is frequently expensive and opaque is largely incompatible with open source.

To address these challenges the Space ROS has worked to develop a requirements management process that balances the need for requirements management with the ease of maintenance and contribution while maintaining compatibility with open source tools. The resulting process, illustrated in Figure 8, leverages two open source requirements management tools Doorstop and FRET. Doorstop is a tool for managing requirements and traceability in a Git based workflow. [6] FRET, or the Formal Requirements Elicitation Tool, was developed by NASA Ames Research Center to define well-structured and testable requirements using a restricted set of natural language, named FRETISH. [7] Doorstop allows users to generate and submit requirements to the Space ROS Git repositories that can be reviewed and accepted via pull requests. It also allows for the generation or reports for requirements traceability. The requirements managed by Doorstop can then be synchronized with FRET using tooling developed by the Space ROS team. FRET allows for the validation of requirements that are written in FRETISH to be validated and for the generation of source code tests based on those requirements. These tests can then be added to Space ROS applications to enable real time verification of software performance.

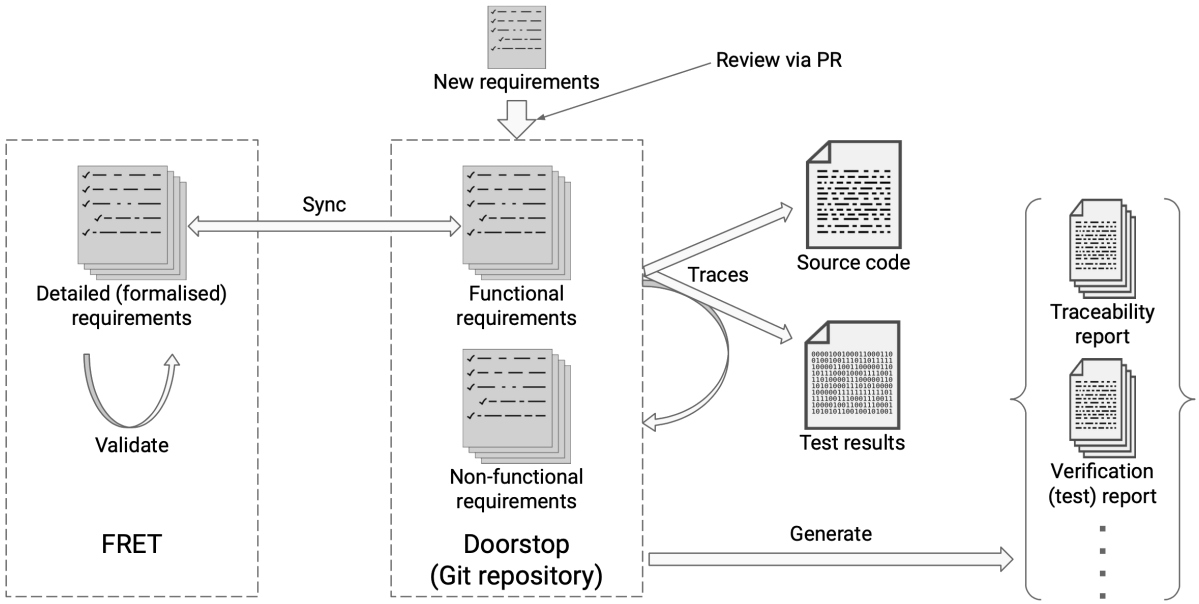

Figure 8: Space ROS Git Based Requirements Workflow using FRET and Doorstop

To help bring the Space ROS codebase and the code for desired packages in line with the appropriate standards the Space ROS team has implemented a static analysis stage into the build system. Static analysis increases code quality by identifying potential errors and provides artifacts related to code coverage and modified condition/decision coverage (MC/DC) testing supporting V&V efforts. Currently two open source static analysis tools developed by NASA have been implemented: Cobra that was originally developed for JPL and IKOS (Inference Kernel for Open Static Analyzers) originally developed by ARC. [8] [9] Cobra provides static analysis capability that works well for

large code bases with fast analysis of general code patterns, common coding flaws, or coding rule compliance. IKOS is a static analysis framework, based on the Theory of Abstract Interpretation, that utilizes LLVM for detecting and proving the absence of runtime errors in C and C++ programs.

Both tools have the capability to output the results of their static analysis tools generate SARIF output, A JSON-based exchange format for the output of static analysis tools. [10] The team has added a filtering capability to avoid duplicate issues for the analysis that result from combining multiple static analysis tools. Currently, the system is capable of removing identical issues, and adding the capability to remove semantic equivalent warnings and errors is on the roadmap. In order to improve the developer experience working with the results of the static analysis tools the team has developed the Space ROS Dashboard and SARIF viewer. [11] This extension, a screenshot of which can be seen in Figure 9, provides insight into static analysis, code coverage, build status, issue burndown.

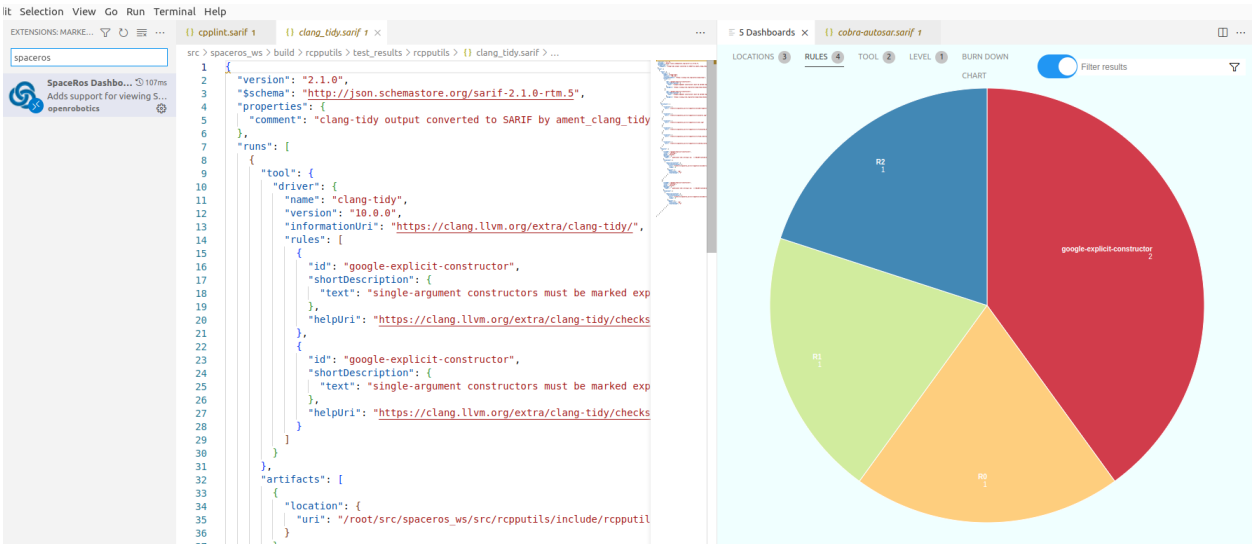

Figure 9: Space ROS SARIF Result Dashboard

3. Additional Verification and Validation Planning

In addition to all the tooling described above the Space ROS team has worked to develop a set of V&V plans that will be released for Space ROS. The objective of these plans is to lay out a path for the further development of requirements, testing and validation for Space ROS to streamline the certification process for Space ROS if it was selected as the infrastructure for a future mission. The certification process is time consuming and requires significant investment from a mission. These preexisting plans should lower the barrier to selecting Space ROS for missions that are considering options for space based robotic operations.

B. Demonstrations

The Space ROS team has developed an initial set of demonstrations for Space ROS that are available in the Space ROS GitHub repository. [12] These demonstrations correspond to the Orbital Station Unpressurized Robotics and Resource Prospecting DRAs. These demonstrations illustrate Space ROS and ROS2 compatible packages being used for space robotics applications and illustrates the compatibility with traditional ROS tooling such as Gazebo. [13]

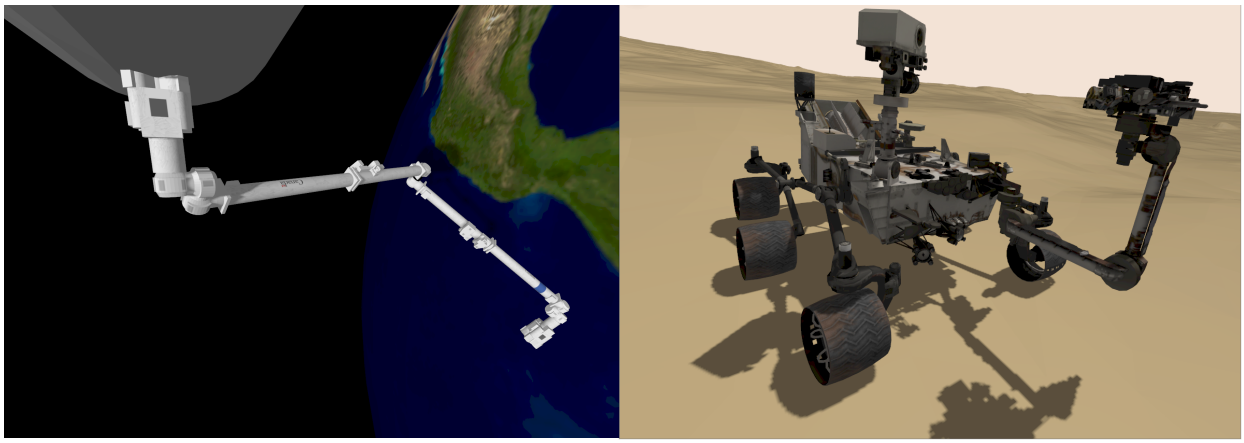

Figure 10: Space ROS Demos based on the Orbital Station Unpressurized Robotics and Resource Prospecting DRAs with Canada Arm and Perseverance as Demonstration Platforms

V. Benchmarking

A. Benchmarking Scope

To quantify the performance of Space ROS when applied to more complex tests that are representative of full space-relevant mission applications Space ROS has been deployed on the Distributed Spacecraft Autonomy (DSA) testbed for scalability testing and benchmarking at ARC. This testbed, shown in Figure 11 is made up of multiple networked units of flight-like platforms such as the Unibap iX5-100, the Diligent ZedBoard (a development platform for the Zynq 7020 which is used in flight processors such as the Xiphos Q7), and the NVIDA Jetson TX2. An existing code base that will be deployed as part of a DSA experiment on the upcoming NASA Starling mission was selected as a starting point for this benchmarking effort. The Starling mission is a swarm of four spacecraft that are sharing data to compute Total Electron Count (TEC) statistics from GPS data, making it a characteristic space-based distributed processing challenge for implementation with Space ROS. [14]

The DSA applications were originally implemented using NASA's cFS as a software framework and RTI's Connext DDS Micro for communications and designed to test the scalability of distributed networking in space by taking TEC data, sharing it between spacecraft through a Comm Application, and combining the data in an Autonomy application that generates plans form monitoring GPS performance. The data flow for the DSA applications is shown in Figure 12. [15] The Space ROS implementation of the system is substantially simpler because it can utilize the built in DDS functionality of Space ROS and consequently doesn't require a Comm app. We plan to characterize the performance parameters described in Table 2. Sample results showing memory and CPU usage from the benchmarking are shown in Figure 13 and Figure 14.

Figure 11: NASA ARC DSA Testbed

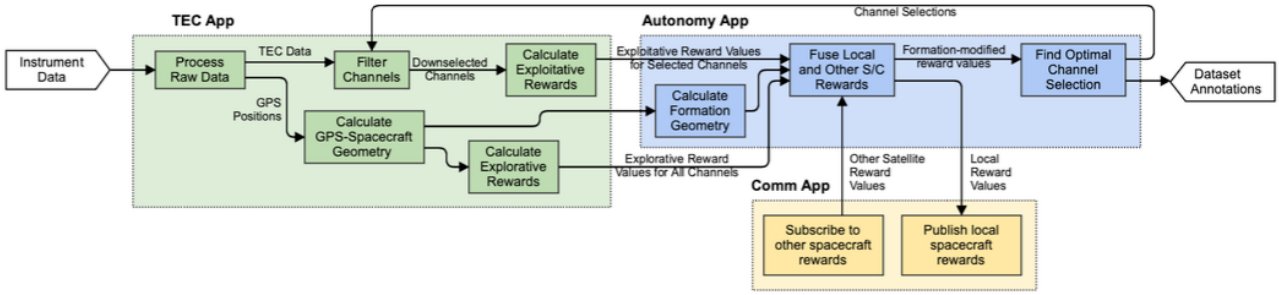

Figure 12: Data Flow for the Distribute Space Autonomy Applications

Table 2: Summary of DSA Metrics to be Collected for Benchmarking

Test Description	Metric Recorded	Metric Details
Message passing tests using but a fully connected and linear mesh network with: Ideal, 10% loss, 50% loss, and 90% loss, 100 ms latency and 500 ms latency network configurations	Deployed Application Size	DSA Apps + Space ROS Compiled File Size
	CPU Utilization	Space ROS CPU Utilization, max, min, mean, variance
	Memory Usage	Space ROS Memory Utilization, max, min, mean, variance
	Topic Latency	Latency between nodes reward and assignment messages, max, min, mean, variance
	Inter Container Latency	Latency between AUTO nodes Auto Sat State Message
	Inter Container Packet Loss	Number of AUTO Sat State Messages not received.
	Total State Messages Ignored	When a message is delayed to long it is just ignored so this combined both dropped packets and ones whose latency was too large.

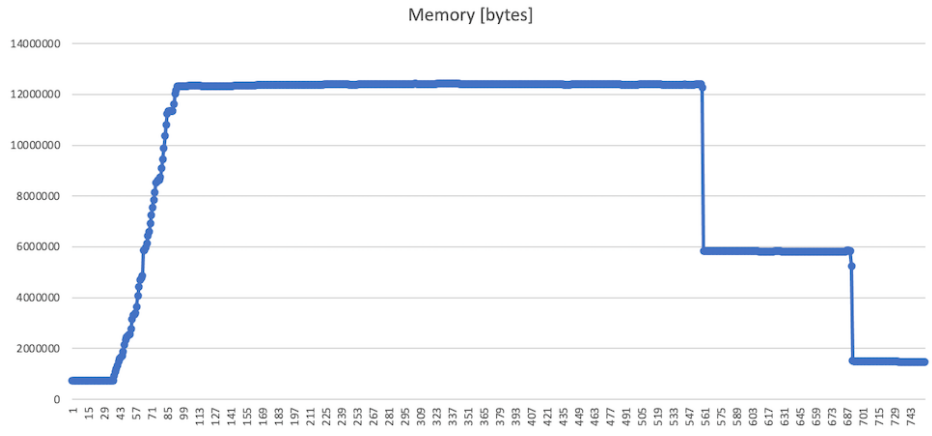

Figure 13: Memory Usage in Bytes of Space ROS Benchmarking Node

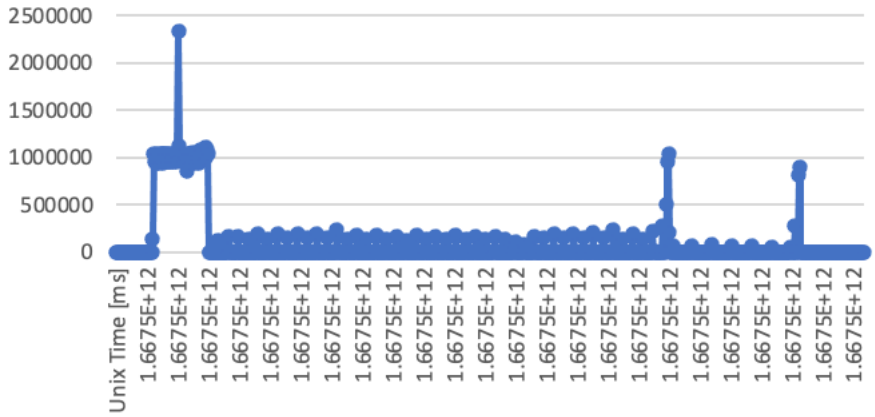

Figure 14: CPU Usage in Hz of Space ROS Benchmarking Node

We are also working to develop an open source community benchmark. The objective of this benchmark will be to answer the questions "Will Space ROS run on my processor?" and "How will Space ROS perform form my application?". We plan to publish this benchmark and the tolling to run it and publish the results to the Space ROS GitHub repository. The Space ROS team will solicit and aggregate benchmarking tests from the community and publish the results on the Space ROS website.

VI. Conclusion

A. Summary

Space ROS will provide an open-source reliable and reusable software framework for the development of new space robotic applications. Adoption of this framework will accelerate the development and deployment of new robotic applications for space and reduce the full life-cycle costs of future robotic systems. The Space ROS team has made substantial progress in enhancing ROS2 to build an initial version of Space ROS that is now publicly available. This includes new features such as enhanced memory safety, new tooling for requirements management and static analysis, and demonstrations of the system applied to characteristic applications. We have provided initial benchmarking data that we will seek to expand in the coming months.

B. Space ROS Community Outreach

The Space ROS team is working with Open Robotics to develop community standards and governance guidelines to allow for more members of the space robotics and flight software communities to get involved with the project. If you are interested in being involved, please reach out to the authors. If you are interested in contributing, we have open sourced the code and will be accepting pull requests.

References

[1] National Aeronautics and Space Administration, "NASA Procedural Requirements," 3 August 2017. [Online]. Available: https://www.nasa.gov/offices/ogc/general_law/npd19009a.html. [Accessed 1 May 2022].

[2] W. Bryan, "2020 NASA Technology Taxonomy," National Aeronautics and Space Administration, 11 MARCH 2021. [Online]. Available: https://www.nasa.gov/offices/oct/taxonomy/index.html. [Accessed 1 MAY 2022].

[3] N. M. Josuttis, C++ 17: The Complete Guide, Nicolai Josuttis, 2019.

[4] D. Thomas, "ROS 2 middleware interface," Open Source Robotics Foundation, Inc., 2019. [Online]. Available: https://design.ros2.org/articles/ros_middleware_interface.html.

[5] eProsima, "MemoryManagementPolicy," 2019. [Online]. Available: https://fast-dds.docs.eprosima.com/en/latest/fastdds/api_reference/rtps/resources/MemoryManagementPolicy.html.

[6] J. Browning and R. Adams, "Doorstop: Text-Based Requirements Management Using Version Control," *Journal of Software Engineering and Applications,* no. 7, pp. 187-194, 2014.

[7] D. Giannakopoulou, A. Mavridou, J. Rhein, T. Pressburger, J. Schumann and N. Shi, "Formal requirements elicitation with FRET," in *International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020).*, 2020.

[8] G. J. Holzmann, "Cobra: fast structural code checking (keynote)," in *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software*, 2017.

[9] G. Brat, J. A. Navas, N. Shi and A. Venet, "IKOS: a Framework for Static Analysis based on Abstract Interpretation," in *nternational Conference on Software Engineering and Formal Methods*, 2014.

[10] OASIS, "Static Analysis Results Interchange Format (SARIF) Version 2.0," OASIS Open, 2019. [Online]. Available: https://docs.oasis-open.org/sarif/sarif/v2.0/sarif-v2.0.html.

[11] Space ROS, "SARIF Viewer for Visual Studio Code," Github, Inc., 2022. [Online]. Available: https://github.com/space-ros/dashboard.

[12] Space ROS, "Space ROS Demos," 2022. [Online]. Available: https://github.com/space-ros/demos.

[13] Open Robotics, "GAZEBO," 2022. [Online]. Available: https://gazebosim.org/home.

[14] L. Hall, "What is Starling?," NASA, 3 August 2022. [Online]. Available: https://www.nasa.gov/directorates/spacetech/small_spacecraft/starling/.

[15] N. C. J. F. Daniel Cellucci, "Distributed Spacecraft Autonomy," in *AIAA Ascend*, Virtual, 2020.